INTERFERENCE ANALYSIS OF SOFTWARE SYSTEMS

James O. Wilder

The Boeing Company
Space and Communications Division
Colorado Springs, CO 80910

Abstract

This paper describes a method for anticipating the occurrence of software faults in terms of a theory of dynamic strength whose functional representation has identical properties with the reliability function. Standard software complexity metrics are assessed to produce a probability distribution representing static complexity, with size as the variate, which is transformed into a complementary static strength distribution. The executing software system is also represented as a probability distribution in terms of size. With the strength and execution distributions expressed in terms of the same variate, analysis of the interference region of the distributions is performed to derive an expression for dynamic system strength. A prerequisite to the interference analysis is to properly characterize the strength distribution at various execution levels. This is accomplished by using another distribution called the return period, which is the average period of time between initializing the execution of a system and the occurrence of a failure.

Introduction

Real-time, reactive software systems interact with and respond to external stimuli during execution. These embedded software systems rely upon test and metrics as early indicators of reliability; yet these approaches can be counterproductive when considered in isolation. An integrated approach that considers the relationships between external stimuli and measurable software complexity attributes could provide higher quality software than relying solely upon development process controls and limited test case behavior results. This paper introduces the concept of using probability distributions derived from the metrics and validated by testing to characterize the behavior of the software in terms of its strength and its execution environment. An expression is derived relating the strength and execution probability distributions. The paper concludes with a discussion of the effects that may occur in their region of intersection, and the possibility of utilizing design safety factors.

Software Metrics

Software metrics can be broadly categorized as either static design attribute metrics or dynamic execution performance metrics.

Static design attribute metrics:

- Software complexity – using McCabe Tools
- Software size – the number of SLOCS (defined as the number of non-commented semi-colons) that comprise the delivered SW products
- Cyclometric complexity – a measure of decision points
- Essential complexity – a measure of un-structuredness
- Number of modules that exceed the cyclometric and essential complexity thresholds

Dynamic performance attribute metrics:

- Resource Utilization – for nominal, stressing, and steady state scenarios against the baselined hardware architecture for the build (CPUs – quantity, speed, and physical memory)
- Memory Utilization - defined as the ratio of system memory used by the process (a.k.a. resident set size) to the physical memory on the machine
- Computer Disk Utilization - defined as the amount of disk storage consumed by the process over time expressed as an absolute value

- Network Utilization - defined as the amount of data sent and received over the network by the process during a given sampling interval, expressed as a rate

Software Integration and Test

Large-scale software developments usually rely upon both development testing and system integration and test to validate the software functionality and assess its quality.

In development testing, the software is rigorously tested beyond its design limits to determine its operational safety margins. System integration and test does not typically impose stressing, design limit excursion scenarios upon the product; rather, the objective here is to assess the software in its installation environment with realistic interfaces and input stimuli.

The objectives of development testing are:

1) To define the subset of all possible test cases that will have the highest probability of detecting the most defects
2) To determine the appropriate or sufficient level of component testing

These objectives are met by:

1) CASE tool quality analyses and assessments of component criticality are used to prioritize development testing activities
2) Error density statistics combined with test coverage statistics are used to determine when sufficient levels of development testing have been achieved.

The goal is to define a test program that supports the overall software defects removal process in an effective and efficient manner, without having to perform exhaustive, comprehensive testing, which is not practically achievable for large real-time embedded software systems.

Accomplishment of this goal may involve both stress testing and external element integration testing. Stress testing – also called excursion testing - is performed to collect data that exercises the performance of the software system at and beyond the edge of the requirements boundary. External element integration testing is performed with the sensor systems and external interfaces to ensure interface and message processing capabilities prior to final software integration.

Stress and Strength Interference Analysis

Can a meaningful assessment of the overall software system performance be made using the combination of the static design and dynamic performance attributes of the software? Or can a more effective use be made of the metrics during the development and test phases, to the point of anticipating the probability the software will encounter faults or failures during execution?

To address these questions, a construct will be developed to apply the theory of stress / strength probability distribution interference analysis. This paper posits that the requirements stress testing data could be used to describe and validate the probability density for system software strength, and external element integration data could be used to describe probability density for stress. The potential intersection of these independent distributions has implications for software product robustness.

Statistical Distributions of Stress and Strength

A fundamental method of explaining failures in reliability theory is to compare the strength of a component with the applied stress. A failure will occur when the stress exceeds component strength. In practical applications of this theory, component strength and applied stress are not characterized as discrete values. For example, there is variability associated with the failure of a metal beam placed under varying loading conditions. The beam will fail largely as a function of load (induced stress), but the inherent variabilities in the material properties (strength) may be due to the structure of the metal or environmental temperature, which can influence the material properties. Similarly, a software system will fail as a function of the stimuli applied to it, and the latent variabilities in the software modules that comprise the system (strength) may be due to the fault exposure frequency induced by stress stimuli.

Strength and stress are therefore often represented as probability distributions. Failures will occur in the region of intersection of these stress and strength distributions, when the stress exceeds the some critical or threshold strength value. Figure 1. illustrates this concept. The probability of success of the system operating under a spectrum of stressing stimuli may be characterized as the probability that the stress does not exceed the strength in random observations from both distributions.

Assuming that probability distributions can be defined that characterize system stress and software strength, this concept of stress / strength probability intersection could also be employed as a technique for assessing software performance. The stress and strength distributions are independent, as program execution depends upon input sequences that do not alter the strength distribution of the system.

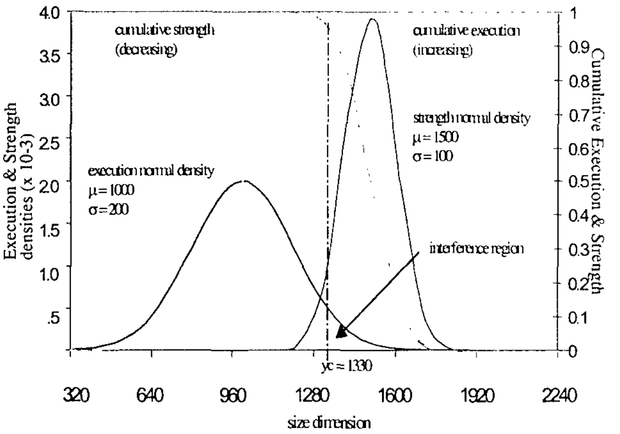

Figure 1. Stress / strength interference

The mean and standard deviation of the execution profile are characterized in terms of the size variate that is common with the size variate of the program strength density. Faults will likely occur with some statistically observable frequency when the probability distribution representing software execution intersects the probability distribution representing static software strength. That is, those failures that occur due to faults in the software will occur exclusively in the execution-strength interference region.

The Statistical Distribution of Software Stress

The "stress" acting on the software system is its dynamic execution sequence in response to input stimuli. This input stimuli spectrum could be defined during the external element integration test phase, when the functional capabilities of the software are run against external event stimuli. The concept of the operational profile (see Figure 2) is well-known[1] and is related to this concept of induced stress. Some modules may be required for all modes, whereas other modules may be rarely encountered, as, for example, modules only invoked during exception handing conditions. Any system of moderate complexity is likely to have at least seven modules, any one of which might be invoked at any time in response to external inputs, such as real-time events handling. The statistical distribution representing a system's execution sequence will approach the normal distribution if the inputs are random.

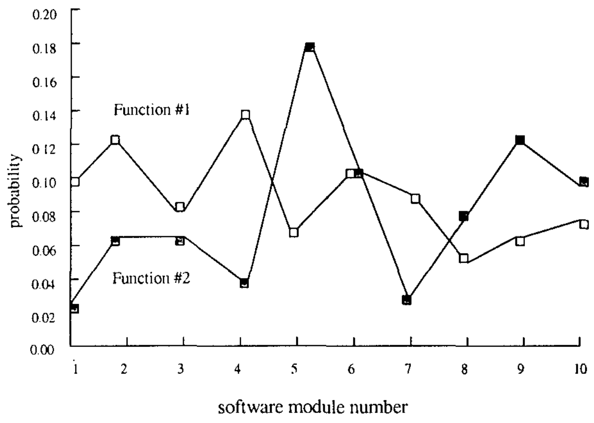

Figure 2. Two operational profiles

For stress / strength interference analysis it is necessary to describe the stress as a probability distribution with some consistent dimension of size as the variate, so we shall consider the probabilities for execution occurrence in terms of module size. Various modules of the software system are invoked for each function or set of functions, and therefore the "size" of the software system, in terms of SLOCS, for example, varies with its execution. When the probability of occurrence of each module is known for each specific function (either through a priori estimation or through event logged data), it is possible to develop a weighted average size for the software function. When all of the potential functions the software is designed to perform are aggregated, the total weighted average size for the system is considered the execution profile. This size can be represented by the normal density function, designated e(x), where x represents a dimension of software size. The mean and variance parameters for the density function e(x) are determined by the weighted averaging. Of course, those systems that execute in a deterministic manner do not necessarily approach the normal form.

An example may help illustrate how a statistical distribution representing the stress of a software system may be characterized:

Mode	Relative frequency	A_i (Size)
1	20	55,000
2	16	45,000
3	14	65,000
4	11	35,000
5	6	75,000
6	2	25,000
7	1	85,000

$\overline{A}$, sample average size = 52,857

σ_{n-1}, sample std. dev. = 13,285

Comments:
Mode 1 is most frequent, but not dominant.
Narrow dispersion, $\dfrac{\overline{A}}{\sigma_{n-1}} = 3.98$

Notes:

1. A_i is the cumulative size of the software system executing the identified system mode. Units are in any dimension representing software size, such as KSLOCs.
2. Relative frequency is the expected number of times the identified system mode is invoked on a relative basis. In the system of this Table, for example, mode 1 is operating 20/2 or 10 times more frequently than mode 6. No single mode is dominant in this example.

The execution levels are the normal execution rate, expressed in size units as some multiple of source lines of code per second, E_1, E_2, E_3,...E_n. In many practical cases, these levels may be identical. Each execution level has a normal execution time t_1, t_2, t_3,...t_n, which is the normal module execution time for that module, assuming no delays or failures due to calls and returns the module may invoke. The expected value of the complete set of execution levels is represented by E_{exp}, where $E_{exp} = \int_a^b e(x) * E^* t \, dx$; a and b are the extremes for the entire defined range of execution, and e(x) is the probability density function for execution as a function of x, which is the dimension of program size.

As an example, if x is executable instructions, then the rate of program execution is given in KIPS (thousands of instructions per second). x might also be lines of code (LOC), and execution in terms of KLOC / second. This average value of execution is not, in general, uniformly distributed throughout the entire program, and it is theoretically desirable to consider the execution level over a differential element of program ΔA. Program execution is then written as

$$e(x) = \lim_{\Delta A \to 0} \frac{\Delta E_{avg}}{\Delta A}$$

e(x) is a continuously differentiable density function for the execution profile, with cumulative distribution E(0) = 0 and $E(x_{max}) = 1$.

The Statistical Distribution of Software Strength

Munson[2] has described the methodology of defining the relative complexity metric, ρ, as a weighted sum of the set of uncorrelated attribute domain metrics. The software complexity metric represents the results of the principal components analysis, also known as factor analysis, of all modules in the software system. ρ represents all raw metrics for in proportion to their contribution of unique variation. The information represented by the original metrics is reduced to an associated complexity metric ρ_j that is comprised of three orthogonal attribute domains, e.g. size, structure, and control. Each module's complexity is independent of the complexities of the other modules within the system.

The composite complexity of the set of modules comprising the software system is the sum of individual module complexities:

$$\overline{\rho_S} = \sum_{j=1}^{m} \rho_j$$

The precision of the static complexity metric applicable to the system level will increase with the number of modules when these modules are combined to characterize the entire system as a statistical distribution.

Software strength may be thought of as a complement of software complexity; both dimensions include the common variate of size, which is also common with the execution distribution, making the strength distribution a more tractable distribution than the complexity distribution for interference analysis. This statistical distribution ideally could be characterized by combining static design metrics as discussed in the preceding paragraphs. This distribution then must be validated by requirements stress testing, where data is collected that exercises the performance of the software element at and beyond the edge of the requirements boundaries.

Assuming these modules are independent, the composite complexity for a system of, say >7 modules forms a normal distribution that may be represented by a function. As software size is one of the variates that form the complexity distribution, the complexity distribution may be transformed into a related function, software strength, with size, y, as the single variate:

$$s(y) = f(\overline{\rho_s})\frac{\overline{\rho_S}}{A_{exp}} = f(\overline{\rho_S})\left|\frac{\partial\rho}{\partial A}\right|$$

$$\lim A_{exp} \rightarrow 0$$

The initial conditions of the strength distribution are S(y → 0) →1 and S(y →∞)→0, where S(y → 0)=1 (see dashed lines, Figure 1.) represents a state of maximum strength, and S(y →∞) = 0 represents a state of minimum strength.

Determinining Software Strength: A Notional Method

Examples of the experimental determination of the statistical distribution of material strength may be found in [3]. In an analogous manner, the strength distribution s(y) of the software system may be determined by estimating the execution time intervals the software may run before encountering a fault or experiencing a failure. If a series of execution time to fault encounter examinations and tests are made on a number of sets of software modules, each set of a specific composite complexity, the results of these tests may be used to statistically characterize the effects of execution levels on the system. Higher levels of execution, where more software modules are invoked to achieve particular functions, will likely exhibit more frequent fault encounters.

As more empirical data becomes available, it should become possible to associate the composite static complexity with the probability (and associated confidence level) that the software system will contain one or more latent faults that would be manifested in software executions. It is also quite likely that software of low composite complexity may not have latent faults.

Let ΔT be the amount of execution time for software of some composite complexity level to encounter a latent fault, which may or may not be manifested as a system failure. For conservatism as well as for practical purposes, there is no attempt to discriminate between harmless faults and those faults that may produce a system failure. The software system (either a single module or set of modules) undergoing examination exhibits a unique execution / fault encounter curve. For any population of modules in such a testing situation, there will be a set of nonintersecting execution / fault encounter curves which can be determined by assessing the spread of ΔTs associated with the execution profile. The average execution / failure curve can then be fitted to the ΔT points from the test data using the least-squares method. Figure 3 illustrates the concept of an execution / strength diagram relating these distributions.

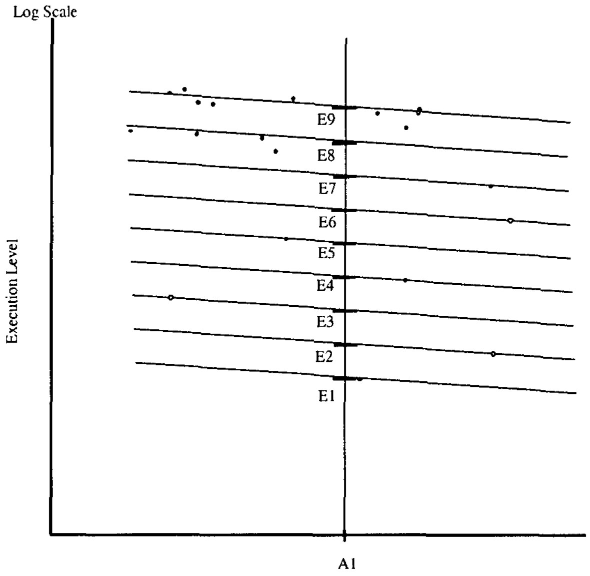

Figure 3. Stress / strength distribution interference

Passing through each test point, execution / failure curves parallel to the average curve are drawn. These will make a family of execution curves. These points of intersections E1, E2,... represent a sample from the strength distribution at the desired execution level. Data from the curves is used to characterize the parameters of the strength distribution.

The execution / failure data for a software system executing a particular function are directly related to the composite complexity of the software modules invoked in performing that function. With sufficient data, it will be possible to describe execution / fault encounter behavior as a function of composite complexity values for the software involved in the execution.

Dynamic Software Strength

Composite strength is inversely related both to composite system complexity and the number of modules (the size of the system) the system invokes in response to input stimuli. If a graph is plotted such that program strength is the ordinate and software size (cumulative source instructions) is the abscissa, then the graph will show program strength decreasing up to some limit, past with the strength has no meaning. Past such a limit it is nearly certain faults will be encountered during execution. Ideally the system should be designed such that its strength distribution does not overlap the intended execution normal distribution. However, when the execution function is known to intersect the strength function, a region of interference of the two distributions is created, as depicted in Figure 1. The intersection of these curves is the point of criticality, y_c past which the strength distribution is below the execution distribution. Failures will occur when $y > y_c$ for both execution and strength in the region of interference.

The probability, designated $\Phi(x)$, that the execution level is not greater than a critical level y_c is written as:

$$\Phi\ (x) = \int_0^{xc} s(y)dy \qquad (1)$$

where the variable x is the same scale of software size as y, and is used to differentiate the variables in the limit and the integrand.

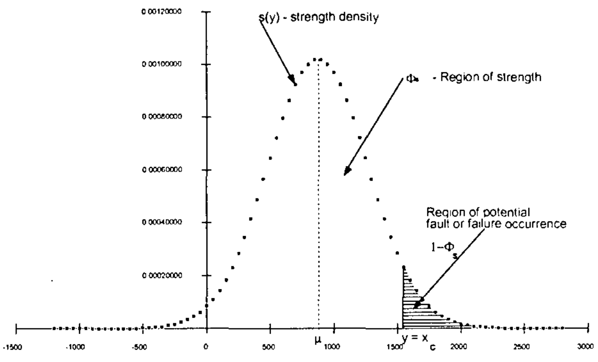

Figure 4. Strength distribution

The corresponding probability the level is greater than y_c is 1 - Φ (x), which is a region where the complexity level is high enough that statistically one or more latent failure producing faults can be expected to be found in the software.

In differential form, the joint density function for execution and strength is

$$e(x)s(y)dxdy \qquad (2)$$

System dynamic strength, designated Φ_s, is the expected value of the strength probability function under the system's execution profile. Φ_s has properties identical to the reliability function. The expected value of Φ (x) operating under the execution profile density e(x) is:

$$\Phi_S\ (y) = \int_{\alpha}^{\omega} e(x_C)\ [\ \int_{-\infty}^{f\ (x)=x_C} s(y)\ dy\ ]dx \qquad (3)$$

The limits α and ω represent the minimum and maximum for the defined range of the execution density function.

This equation, conceptually illustrated in Figure 1, represents the expected value of the strength function up to a critical level when operating under the execution distribution. This is a general relationship between the execution profile distribution and the strength distribution for the system under analysis, and is independent of the types of the distributions.

The Statistical Distribution of the Critical Value

As stated in the introduction, both the cyclomatic and essential complexity are static design attribute metrics which have associated critical or threshold values. However, at the element level, when combinations of modules are executing, the critical size value is not a discrete value under dynamic conditions, and may be randomly distributed. Statistics provides a distribution to describe the distribution of the critical size value: the return period[4]. The return period is the average time interval for the system to exceed the value of x_C. Mathematically, the return period is expressed as:

$$T(x) = \Delta T\ /\ 1\ -\ \Phi\ (x)$$

The return period, determined from the experimental procedure described previously, is a

Rayleigh distribution that characterizes the instantaneous peak values of a Gaussian process executing at a constant frequency. (See Figure 5, Normal execution with Rayleigh distributed peak values.) The system is successful so long as all the peaks are below the critical size level.

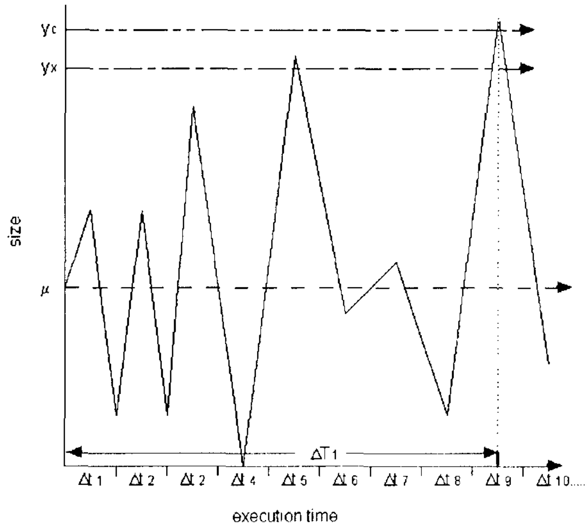

Figure 5. Software exceeding critical strength level The software, executing at a constant frequency about a mean value, μ, encounters modules which exceed the critical strength level according to a Rayleigh distribution.

Discussion

The construct developed herein can be extended to apply safety factors in the design process, and to utilize the return period as the upper limit of the integral for f(x) in equation (3). These extensions are discussed briefly below.

Safety Factors

In the joint density function for execution and strength (equation (2)), the case where the execution level exceeds the strength is $x - y$, represented by the new variate z, where $z = x - y$. Through a transformation of the variate, the joint density is

$$e(x)s(x - z)dxdz,$$

and the density function of z, g(z), is expressed as

$$g(z)dz = \int_{\alpha}^{\omega} e(x)s(x\text{-}z)\, dx\, dz.$$

With the strength distribution appropriately characterized, the failure potential is the probability that a randomly activated set of software code, as determined by the execution profile, is of insufficient strength due to complexity levels that exceed the threshold for fault-free execution.

If the distributions are normal, the average and standard deviation of the difference distribution are

$$\mu_{diff} = \mu_e - \mu_s$$

$$\sigma_{diff} = \sqrt{\sigma_e^2 + \sigma_s^2}$$

where μ and σ represent average and standard deviation and subscripts e and s denote execution and strength. The probability of failure is

$$Z = \mu_{diff} / \sigma_{diff}$$

where Z is the standardized normal variate that provides the probability that the difference is less than zero, a failure state.

Safety factors could be employed to develop software that is less subject to failure during periods of high execution, or when failures are least tolerable in real-time applications. Safety factors may be designed to specify an execution level at some number of standard deviations of e(x) above the mean execution, μ_e. The safety factor can be utilized as a verification tool, after the system has been tested; early in the software design phase, both the form and the standard deviation of the strength and execution distributions can be assumed until empirical evidence provides realistic parameters.

The Return Period

Analysis of the region of interference of these distributions can be performed when the strength distribution is defined in terms of its fault or failure encounter behavior at various execution levels. This is done experimentally by the return period, which is the average period of time between initializing the execution of a system and the occurrence of a fault or failure. The return period itself can be expressed as a statistical function if empirical data is available to establish the behavior of return periods for a number of software systems. The relationship between execution levels and strength distributions may be described by using the return period as the upper limit of the integral of the strength distribution.

Summary

The relative complexity metric represents the static complexity with a single numerical value. Each module in a software system is of a discrete size, measured in lines of code or any other appropriate, consistently applied dimension of size, and has an associated relative software complexity metric. The underlying distribution of relative complexity metrics for the system of modules approaches the normal form as more modules are considered. By means of a transformation of the variate, this distribution is redefined as a strength distribution (composite static strength) for the software system. A second distribution that may be characterized in terms of size is the execution profile. The strength of a software system is a function of its execution profile and composite static strength.

Software metrics are often presently utilized to a limited extent in software development and test. Systematically estimating and deriving the metrics through the development and test processes could serve both to validate the metrics and to quantify the reliability and operational margins of the software product.

References

[1] J.D. Musa, Operational Profiles in Software-Reliability Engineering, IEEE Software, March 1993, pp. 14-32.

[2] J. C. Munson, The Use of Software Metrics in Software Reliability Modeling, Conference on Software Reliability and Safety for the '90s, Nov. 1993

[3] Kapur, K.C. and Lamberson, L.R. , *Reliability In Engineering Design*, New York: John Wiley and Sons, Inc., 1977

[4] Gumbel, E.J., *Statistics of Extremes*, Columbia University Press, New York, 1958, pp.21-26